Blumengarten

Tassilo Tanneberger

November 24, 2019

1 Ideen der Algorithmen

Ist eine Sammlung an prinzipiellen Gedanken, die ich mir gemacht hatte, bevor ich angefangen habe zu implementieren.

1.1 Bewertung von Farben

w_i ... Wichtung der Preferenz
w_f ... Finale Wichtung der Farbe
N ... Anzahl der Preferenzen mit dieser Farbe

$$w_f = \sum_{i=1}^{N} (w_i^2 + 1) \tag{1}$$

Diese Gleichung für die Wichtung hat den Vorteil, dass durch das Quadrat, die Wichtung des Nutzers mehr gewertet wird, als wie oft er diese Farbe angab.

1.2 Priotiry Places

Jeder Platz auf dem Blumenbeet hat eine gewisse Anzahl an Nachbarblumen n. Wobei n genau den Wert dieses Platzes repäsentiert. Dadurch, dass die Anordung auch bis zu einem gewissen Grad symmetrisch ist, kann man nun beginnen, die "wichtigen" Farben an wichtige Plätze zu packen. Somit ist mehr oder weniger eine feste Reihenfolge der Platzierungen gegeben, wo zuerst die wichtigsten Farben (mit dem höchsten w_f) gesetzt werden.

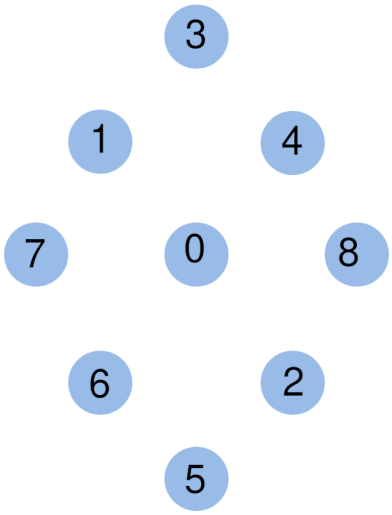

Figure 1: Reinfolge in der die Plätze vergeben werden

Wobei hier zu beachten ist , dass 0, 1, 2 immer mit den am höchsten gewichteten Werten platziert werden. Auch wenn alle 7 Farben genutzt werden sollen, bleiben die $2 = 9 - 7$ für 1 und 2 übrig.

2 Implementierung

Alles wurde einfach aus pragmatischen Gründen in Python3.7 implementiert. Python als Highlevel Sprache ist einfach ideal für solche kleine Projekte , weil man sich z.B nicht um Memory Leaks, nervende Seq. Faults und Ähnliches kümmern muss.

2.1 Sorted by Priority

2.1.1 User Input

Ich habe den Nutzer-Input etwas anders designed, indem ich zuerst die Preferenzen angeben lasse und danach frage, ob er noch andere Farben möchte. Das macht mehr Sinn, weil man damit Konflike vermeiden kann. Ein Beispiel - der Nutzer sagt, er möchte 2 Farben haben und sagt dann Rot-Blau-3 und Grün-Blau-2 und er hat einen Konflikt erzeugt , da er 3 Farben in den Preferenzen angibt, an Stelle von vor 2.

2.1.2 Bewertung der Farben

Implementierung der Bewertung.

In Datei ./Blumengarten/Field.py

```
def __calcualate_value(self):

    self.colour_values: List[int] = [0] * 7

    # Element: 0 First Colour, 1 Second Colour, 2 Weight

    for element in self.user_input.preferences:

        self.colour_values[element[0]] += element[2] ** 2 + 1
        self.colour_values[element[1]] += element[2] ** 2 + 1
```

Wobei "preferences" eine Liste mit Tupeln der Form (color1, color2, weight) ist.

2.1.3 Generieren der anderen Farben

Diese Methode erzeugt die zusätzlichen Farben zufällig. Zufällig aus dem Grund, um noch ein wenig Variation ins Spiel zu bringen.

```
def generate_random_colors(self):
        self.additional_random: List[int] = []
        self.final_colors: List[int] = self.user_input.used_colors
        if self.user_input.colors_count > 0:
        for x in range(self.user_input.colors_count):
                ran: int = random.choice(self.user_input.open_colors)
                del self.user_input.open_colors[self.user_input.open_colors.inde
                self.additional_random.append(ran)
```

2.1.4 Platzieren

Zuerst werden die oben angesprochenen freien besten Plätze statisch mit den besten Farben besetzt, wie in der Reihenfolge in Figure1 beschrieben, siehe

Funktion placespare(). Wenn dieser Prozess abgeschlossen ist, müssen wir alle übrigen Farben noch unterbringen. Übrige Farben sind nun alle zufälligen und nicht die besten zwei. Von diesen Farben wird es jeweils nur eine geben. Da unser Ziel die Maximierung der Anzahl der Punkte ist, müssen die restlichen Blumen so gesetzt werden, dass die Synergien besonders gewichtet werden. Die Platzierung wird nach folgender Schrittfolge getätigt:

1. Indices aller umliegenden Plätze herausfinden
2. Nach den Farben schauen, die sich auf diesen Plätzen befinden
3. Dann wird die Farbe mit der höchsten Synergie mit umliegenden Farben ermittelt
4. Farbe wird ins Feld geschrieben und aus den verfügbaren Farben heraus gestrichen

Synergie Funktion

```
def query_highest_syn(self, color: list, already_used: list, possible_colors
    ratings: List[int] = [0] * 7

    for x in possible_colors:
        if x in self.user_input.used_colors and x not in already_used:
            for c in color:
                ratings[x] += Field.get(self.field.prepared, (c, x))

    if ratings == [0] * 7:
        i: int = random.choice(possible_colors)
    else:
        i: int = ratings.index(max(ratings))
    return i
```

Das Programm iteriert durch die verschiedenen Möglichkeiten. Weil es sehr übersichtlich viele Farben sind ist das nicht gerade besonders aufwändig. Es schaut, ob die Farbe noch zur Verfügung steht - also in possiblecolors gelisted ist und nicht usedcolors ist. Dann schaut das Programm sich die umliegenden Farben (color) an und fragt die Wichtung zwischen diesen beiden Farben ab. Wenn es keine gibt, ist der Wert einfach 0. Zum Schluss müssen wir dann noch nachsehen, ob es überhaupt Synergien gab. Wenn nicht wird eine zufällige Farbe gewählt. Sonst der Wert, der insgesamt die höchsten Synergien erreichte.